Домен:

Innotter - аналог твиттера со своими плюшками.

Сущности:

```
from django.db import models
from django.contrib.auth.models import AbstractUser


class User(AbstractUser):
    class Roles(models.TextChoices):
        USER = 'user'
        MODERATOR = 'moderator'
        ADMIN = 'admin'

    email = models.EmailField(unique=True)
    image_s3_path = models.CharField(max_length=200, null=True, blank=True)
    role = models.CharField(max_length=9, choices=Roles.choices)

    title = models.CharField(max_length=80)
    is_blocked = models.BooleanField(default=False)


class Tag(models.Model):
    name = models.CharField(max_length=30, unique=True)


class Page(models.Model):
    name = models.CharField(max_length=80)
    uuid = models.CharField(max_length=30, unique=True)
    description = models.TextField()
    tags = models.ManyToManyField('innotter.Tag', related_name='pages')

    owner = models.ForeignKey('innotter.User', on_delete=models.CASCADE,
related_name='pages')
    followers = models.ManyToManyField('innotter.User',
related_name='follows')

    image = models.URLField(null=True, blank=True)

    is_private = models.BooleanField(default=False)
    follow_requests = models.ManyToManyField('innotter.User',
related_name='requests')

    unblock_date = models.DateTimeField(null=True, blank=True)
```

```
class Post(models.Model):
   page = models.ForeignKey(Page, on_delete=models.CASCADE,
related_name='posts')
   content = models.CharField(max_length=180)

   reply_to = models.ForeignKey('innotter.Post', on_delete=models.SET_NULL,
null=True, related_name='replies')

   created_at = models.DateTimeField(auto_now_add=True)
   updated_at = models.DateTimeField(auto_now=True)
```

Дисклеймер: сущности являются примерными, допустимы доработки и улучшения. Так же некоторые из них стоит вынести в отдельные аппки, например User

Нефункциональные требования:

- Общее:
  - UNIX подобная система (Ubuntu предпочтителен)
  - Работа по GitHub Flow
    - две основных ветки (master, dev)
    - ответвляемся всегда от dev в feature-ветки
    - делаем работу в них и пушим
    - делаем pull request в dev
    - когда сделали pull request, пишем ментору, приступаем к новой задаче не дожидаясь ревью
    - если ментор оставит комментарии на pr, исправляем и заливаем изменения
    - если ментор аппрувит pr то заливаем его в dev
  - PostgreSQL в качестве бд для core app
  - AWS DynamoDB для микросервиса
  - Для хранения файлов использовать AWS S3
  - Для отправки email использовать AWS SES
  - Две разные очереди RabbitMQ для микросервиса и Celery
- Python:
  - Common:
    - Последняя стабильная версия
    - Для управления виртуальным окружением использовать pipenv **или** poetry
    - Следовать PEP 8 (русская версия)
    - Связь между микросервисом и основной аппкой через RabbitMQ
    - Для взаимодействия с AWS использовать boto3
    - Валидация JWT-токена в обоих приложениях
    - Вся бизнес-логика должна быть покрыта юнит тестами, использовать оба подхода (mock, fixture)

    - Core app:
        - Django + Django Rest Framework
        - Использовать ViewSets + routers
        - Использовать ModelSerializer
        - Celery + RabbitMQ для отправки уведомлений
        - Отделять бизнес-логику приложения от views в отдельные сервисы.
        - Кастомная JWT аутентификация с помощью Middleware (разрешается использовать PyJWT)
    - Microservice:
        - FastAPI
        - Pydantic
        - Readme файл с описанием структуры проекта и команд
- Docker + docker-compose:
    - Точка входа выносится в отдельный файл (entrypoint.sh)
    - Один Dockerfile на Celery и Django, но разные входные точки
    - Отдельный Dockerfile для микросервиса
    - База данных должна быть развернута в docker-compose

Функциональные требования:

У пользователя может быть одна из 3 ролей:

- Администратор
- Модератор
- Юзер

1) Администратор имеет права просматривать любые страницы, блокировать их на любой промежуток времени и перманентно, удалять любые посты и блокировать пользователей. Также администратор имеет доступ к админ панели.
2) Модератор имеет права просматривать любые страницы, блокировать их на любой промежуток времени, удалять любые посты.
3) Пользователь может:
    a) зарегистрироваться, залогиниться
    b) создать страницу, редактировать ее название, uuid и описание, добавлять и удалять к ней теги, делать ее приватной/публичной
    c) подписываться на чужие страницы (кидать запрос на подписку в случае приватных страниц)
    d) смотреть список желающих подписаться на страницу (в случае приватной) и подтверждать/отказывать в подписке (по одному или сразу всем)
    e) писать посты на своих страницах, редактировать их, удалять
    f) лайкать/анлайкать посты, отвечать на них от лица какой-либо из собственных страниц

    g) просмотреть пролайканные посты
4) Аватары пользователей и страниц хранятся в облачном хранилище.

Система должна:

1) Отправлять уведомления подписчикам на email о новых постах.
2) Показывать новостную ленту (все новые посты подписанных и своих страниц)
3) Автоматически блокировать все страницы пользователя, если пользователь заблокирован
4) Предоставлять поиск страниц по названию/uuid/тегу и пользователей по username/имени (с помощью одного эндпоинта)
5) Делать проверку расширений загружаемых файлов
6) Показывать статистику страниц (количество постов, подписчиков, лайков и т.д.), которая формируется на микросервисе, только пользователям-владельцам страниц

Последовательность действий

1) Создать приватный гит репозиторий и дать доступ своему ментору.
2) Создать AWS аккаунт (в связи с тем что доступ может появится через неделю)
    a) Если aws не присылает код подтверждение можно использовать sms receiver online
    b) После конца тестового задания удалите все сервисы на авс, для избежания списывания денег
3) Создание проекта + настройка окружения
    a) Создание django проекта
    b) Установка всех нужных пакетов через poetry или pipenv
    c) вынос всей sensitive data в .env с помощью python-dotenv
4) Docker + docker-compose
5) Модели
6) CRUD + logic + permissions
7) Celery + AWS
8) Microservice

Отчетность

Каждый день утром отписываем ментору следующее:

1) Что сделал вчера
2) Что буду делать сегодня
3) Какие есть трудности

Дедлайны

Что должно быть сделано	К какому моменту
AWS account + repo + environment setup + Docker	5-7 рабочих дней
Models + CRUD	10-12 рабочих дней
Logic + Permissions	15-17 рабочих дней
AWS	20-22 рабочих дня
Celery	25-27 рабочих дней
Microservice	30-32 рабочих дня

P.S. Дедлайны не значат что у вас ровно в день дедлайна должен быть один пулл реквест. Нужно их делать в процессе работы выделяя логически готовые части. На примере 1 недели: лучше сделать пулл реквест на настройку окружения и пулл реквест на докер чем один вместе

Собеседования

После каждой недели нахождения на стажировке вы будете проходить пробные собеседования по технологиям, что изучили, и по резюме, что вы будете составлять на протяжении стажировки.
На первой неделе составьте **на английском** текст про себя и свою профессиональную деятельность, а дальше на протяжении стажировки дополняйте его всеми технологиями, что вы изучили.

Пример*

* - не является жестким примером, старайтесь излагать мысли по своему

Дополнительные задания

1) Django ORM
2) GIT branching
3) AWS

Создать на AWS 2 лямбды:

1) Lamba function с триггером на APIGateway, которая принимает файлы. Проверяет является ли данный файл csv-файлом и сохраняет его на S3
2) Lambda function с триггером на S3 которая валидирует с помощью pandas данные, убирает все нулевые значения и сохраняет данные в DynamoDB

4) SQL